IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s):	Prudvi, et. al.		
Serial No.:	09/212,291	Examiner:	T. Thai
Filing Date:	December 16, 1998	Art Unit:	2186
Title:	Transaction Manager and Cache for Processing Agent		

APPEAL BRIEF

ASSISTANT COMMISSIONER FOR PATENTS
Washington, DC 20231

Sir:

This is an Appeal from an Office Action dated July 2, 2002, finally rejecting each of the pending claims 1-29. The Notice of Appeal was filed on August 28, 2002. The period to file this Appeal Brief expires on March 28, 2003.

The Commissioner is hereby authorized to charge the appeal brief fee of $320.00 to Kenyon & Kenyon Deposit Account No. 11-0600. Concurrently, Appellants request a one month extension of time and authorize the Commissioner to charge payment of $110.00 and any additional fees which may be necessary for consideration of this Appeal.

REAL PARTY IN INTEREST

The real party in interest in this matter is Intel Corporation. *See* Assignment (recorded February 17, 1999 at Reel 9881, Frame 0608).

RELATED APPEALS AND INTERFERENCES

There are no related appeals or interferences.

STATUS OF THE CLAIMS

The application contains claims 1-7, 11-21 and 23-29.

Claims 17-21, 23-24 and 27-29 stand rejected under 35 U.S.C. §102(b) as anticipated by Sachs, et al., U.S. Patent No. 4,884,197 ("Sachs").

Claims 17-21, 23 and 27-28 stand rejected under 35 U.S.C. §102(b) as anticipated by Scales, et al., U.S. Patent No. 4,914,573 ("Scales").

Claims 1-7, 11-16 and 25-26 stand rejected under 35 U.S.C. § 103(a) over Sachs in view of Scales.

STATUS OF THE AMENDMENTS

To date, all substantive amendments to the application have been entered, except for an Amendment After Final that was submitted concurrently with this Appeal Brief. This most recent amendment cancels claims 8-10 and 22 from the application.

SUMMARY OF THE INVENTION

An internal memory cache is a memory bank that acts as a temporary storage area bridging an external memory and a processor or other agent. Cache memory is generally faster than external memory. Therefore, a cache memory allows an agent to access data at higher speeds than would normally be possible without the cache. The larger the cache memory, the faster the overall performance of the agent, since there will be a greater chance that a copy of the required instruction or data element will already be in the cache.

Conventionally, cache memories have exchanged data with external memories in increments of data having a prescribed maximum length. This prescribed maximum length has traditionally been called a "cache line," because it corresponds to the maximum amount of data that a cache memory can store in one "line" as a single addressable unit.

External bus transactions typically have been designed to transfer data in these same cache line-sized increments. A "cache line," therefore, has traditionally referred not only to the amount of data stored within an entry of an agent's cache memory, but it has also referred to the maximum amount of data that could be exchanged in a single bus transaction. In the prior art, there has been a one-to-one relationship between the size of a cache line and the maximum amount of data that can be transferred in a single bus transaction. Thus, until now there has been no need to distinguish between the size of a cache line and the size of a single bus transaction.

To simplify terminology and enable the size of a cache line to be distinguished from the size of a bus transaction, the present invention uses two terms, "data line length" and "cache line length," which are defined as follows:

> **data line length** – the maximum amount of data that can be transferred in a single bus transaction.
>
> **cache line length** – the length of an individual line in an agent's cache memory.

Until the present invention, the relationship between data line length and cache line length had been at most one-to-one. Embodiments of the present invention break this one-to-one relationship by introducing a new ***many-to-one relationship*** of data line length to cache line length. Rather than have an internal cache that is identically sized to the maximum capacity of a single bus transaction, these embodiments provide cache lines that can store several increments of this capacity. Thus, in the present invention, a cache line length is a multiple of data line length.

FIG. 2 of the present invention further illustrates this important many-to-one relationship that is not found in the prior art. In Fig. 2, a single cache line 510 stores ***multiple*** data line lengths 530, 540 of data:

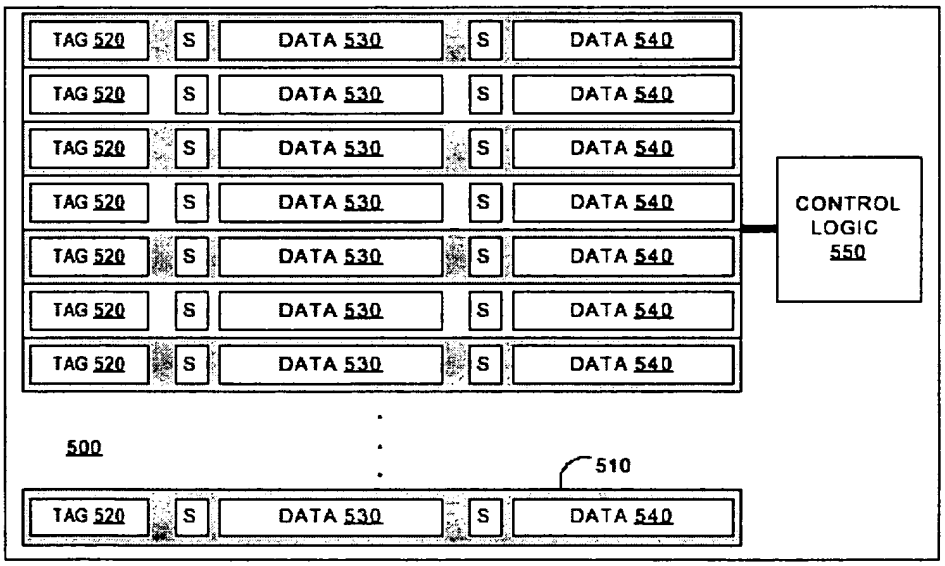


FIG. 2

Storing multiple data line lengths of data provides several advantages. First, cache lines 510 may have a single field 520 to store address information for both data lines 530, 540. This architecture may conserve area when the cache is manufactured in an integrated circuit. Additionally, state information S may be stored for each data line 530, 540 permitting cache coherency operations to occur on a data-line basis.

Claim 1 captures succinctly this important many-to-one relationship that is not shown in any reference cited by the examiner. Claim 1 recites:

> 1. A processing agent to transfer data of a predetermined data line length in an external transaction, the agent comprising an internal cache having a plurality of cache entries, ***each entry sized to store multiple data line lengths of data***.

Other embodiments of the invention provide for a transaction queue to manage information in the cache. Transaction queues generally manage requests on the external bus. According to the present invention, a ***single*** entry of the transaction queue may include fields for ***multiple*** transactions on the external bus. FIG 3, reproduced below, illustrates an embodiment of the transaction queue entry:

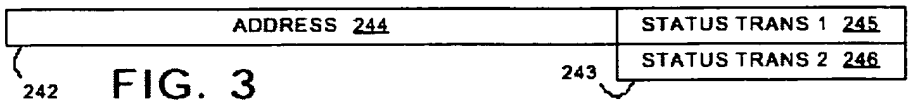


FIG. 3

The transaction queue entry may include an address portion 244 to store address information, a first status field 245 to store data associated with the first transaction and a second status field 246 to store data associated with the second transaction. Thus, when a single data request is loaded into the external transaction queue, the queue may respond by issuing a sequence of transactions addressed not only to the address of the original data request but also to other neighboring addresses.

These embodiments, when used in an agent, provide for a natural prefetch mechanism. Ordinarily, when a data request from the agent's core misses the internal cache, the request propagates to the external bus and a data line-sized unit of data will be supplied to the core and stored in the cache. These embodiments not only retrieve the data line requested by the core, they also retrieve multiple data lines from other memory locations nearby. Since program flow often progresses linearly in the absence of discontinuities, the prefetched data is likely to be used by the core in the future. This prefetch mechanism increases the likelihood that the cache will store data to be used by the core before the core asks for it and, therefore, can increase the agent's performance.

The Examiner rejected all pending claims as anticipated either by Sachs or Scales, or as rendered obvious by their combination. However, neither Sachs nor Scales, separately or in combination, teach or suggest breaking the one-to-one relationship between cache lines and data lines as described above. Therefore, as described more particularly below, the Examiner's cited art is insufficient to anticipate the claimed invention or to render it obvious.

ISSUES

The issues on appeal are:

1. Whether claims 17-21, 23-24 and 27-29 patentably distinguish over Sachs under 35 U.S.C. § 102(b);
2. Whether claims 17-21, 23 and 27-28 patentably distinguish over Scales under 35 U.S.C. § 102(b); and
3. Whether claims 1-7, 11-16 and 25-26 patentably distinguish over Sachs in view of Scales under 35 U.S.C. § 103(a).

GROUPING OF CLAIMS

Claims 1-7 and 25 are directed to a processing agent with an internal cache having cache entries that are sized to store multiple data line lengths of data.

Claims 11-16, 24, 26 and 29 describe a processing agent comprising an internal cache with cache entries sized to store multiple data lines, and a transaction queue system that posts external transactions related to a single data line.

Claims 17-21, 23 and 27-28 are directed to methods of processing a data request comprising posting a series of external data line transactions for a cache line when the data request misses the cache.

A separate basis for patentability exists for each group of claims. However, except to the extent otherwise indicated below, the respective groups of claims do not stand or fall together for purposes of this appeal.

ARGUMENT

Sachs Fails to Teach Every Element of Claims 17-21, 23-24 and 27-29.

The Examiner rejected claims 17-21, 23-24 and 27-29 under 35 U.S.C. § 102(b) as anticipated by Sachs. Sachs does not disclose each and every element of these pending claims, and therefore the Examiner's rejection must be reversed.

Sachs describes a modification to the general *von Neumann* computer architecture, in which a cache memory, interposed between a microprocessor and main memory, is partitioned into an instruction section and a data section, each of which operates independently from the other. Sachs, Abstract; col. 1:40-2:26. Additionally, and most importantly for purposes of comparing Sachs to the

present invention, Sachs also describes a way of improving the efficiency of cache memory by storing multiple data words in the same cache line. Sachs, col. 22:14-48.

The Examiner appears to believe that Sachs' feature of storing multiple data words in one cache line demonstrates a many-to-one relationship between data line length and cache line length. Respectfully, the Examiner is mistaken. The present invention teaches a many-to-one relationship of data ***line*** length to cache line length, not data ***word*** length to cache line length. The Examiner has apparently failed to appreciate that Sachs' external bus transaction protocol can transfer a maximum of *sixteen* 32-bit data words in one transaction. See, e.g., Sachs, col. 10:24-34; see also, col. 14:5-8 ("...the length of the longest [bus] cycle is ... a sixteen word read or write..."). Sachs' data line length is therefore 16 x 32 or 512 bits.

Sachs' cache line length is 128 bits. See, e.g., Sachs, col. 22:45-48. One of Sachs' 128-bit cache lines is completely unable to store even one full 512-bit data line. Thus, the relationship between Sachs' data line length to cache line length is one-to-many, not many-to-one. When contrasted with the present invention's many-to-one relationship, it is clear that Sachs falls short and cannot anticipate the present invention.

Claims 17-21, 23-24 and 27-29 are Allowable Over Sachs.

Claims 17-21, 23-24 and 27-29 of the present invention all recite a cache line that is *sized to store multiple data line lengths of data.* Sachs fails to teach or suggest this feature. The present invention defines a data line length to be the maximum length of data that may be transferred in a single bus transaction. See Application at 1-2. Throughout prosecution, the Examiner has never challenged this definition. Instead, while seeming to ignore Sachs' own definition of a data line length (which makes clear that Sachs teaches *less than* a one-to-one relationship of data line length to cache line length), the Examiner rejected without explanation claims 17-21, 23-24 and 27-29 as being anticipated by Sachs. Final Office Action (paper no. 18), p. 4-5 (failing to explain how Sachs teaches a cache line sized to store multiple data line lengths of data); p. 6 (referring only to Scales and not to Sachs). Appellants respectfully maintain that Sachs fails entirely to teach or suggest a cache line that is sized to store multiple data line lengths of data. Hence, Sachs cannot anticipate these claims.

Figs. 10A and 10B illustrate Sachs' cache memory storage structure. See also, Sachs, col. 22:14-48. Together, these two figures reveal a cache memory system organized into two identical but independent fields W and X, each of which contains 128 lines. Id. A narrow third field, labeled U (on the left side of Fig. 10A), is used to indicate whether W or X was most recently used. Id.

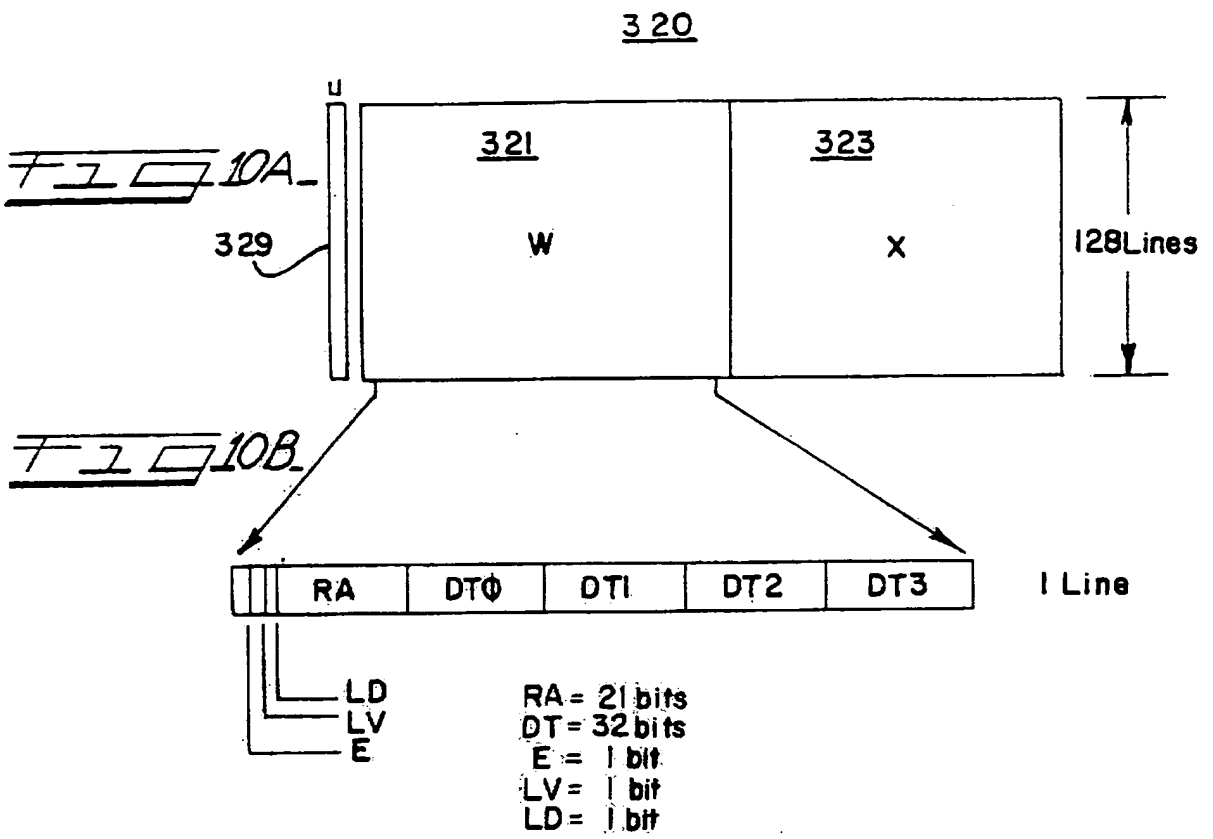


Fig. 10B illustrates the structure of each cache line within the W field. The X field structure is identical. Each line in the cache memory contains several status sub-fields (E, LV and LD), an address sub-field RA, and four data words, DT0, DT1, DT2, and DT3. Because Sachs further defines each data word to be 32 bits, each cache line is capable of storing 4 x 32 or 128 bits of information associated with each memory address RA. See, e.g., Sachs, col. 4:44; col. 5:11; col. 6:47-57; col. 22:47. Sachs refers to this four data word, 128 bit collection as a "quadword." Sachs, col. 23:10-11. Because the W and X fields are not connected, a cache line in the W field functions independently from its corresponding cache line in the X field. They cannot be combined together. Consequently, each of Sachs' cache lines is limited to store a maximum of one quadword. Sachs' cache line length is therefore 128 bits.

Before comparing Sachs' cache memory structure with the present invention, it is necessary to know Sachs' data line length. Recall that the present invention defines a "data line length" to be the maximum amount of data that can be transferred in a single bus transaction.

Sachs bus protocol describes exactly four (4) different quantities of data that may be transferred across the system bus from main memory to cache memory. Sachs, col. 10:24-57. These four quantities are:

(1) a single 32-bit word transfer [col. 10:29];

(2) a 128-bit quadword transfer [col. 10:30];

(3) a **512-bit** sixteen-word transfer [col. 10:31]; and

(4) a cache-MMU single- or partial-word write [col. 10:31; 10:56-57].

The largest of these quantities is a 512-bit sixteen-word transfer. Sachs further explains that this sixteen word transfer is the maximum amount of data that can be transferred on the bus in one transaction. Sachs, col. 14:5-8 ("...the length of the longest [bus] cycle is ... a sixteen word read or write...").

Because Sachs' data line length is 512 bits, each of Sachs' 128-bit cache lines can only store *less than* one data line length. Thus, the relationship between Sachs' data line length and his cache line length is *less than* one-to-one. Even allowing for the possibility that Sachs' 16-word transfer might be a special case, and his quadword transfer (128 bits) should be more properly considered the standard data line length (which is an unjustified assumption), the relationship is still no better than one-to-one.

In contrast, the present invention exceeds this one-to-one relationship. Rather than having an internal cache memory that is sized to match the capacity of a single bus transaction, embodiments of the present invention provide cache entries that can store several increments of this capacity and thus exhibit a many-to-one relationship not disclosed in the prior art.

Independent claims 17 and 23, for example, both describe a method of processing a data request, "wherein each cache line is sized to store multiple data line lengths of data." Independent claim 24 describes a processing agent, comprising "an internal cache having a plurality of cache lines, each cache line including ... a plurality of cache entries, each cache entry sized to store one data line length of data." Because claims 17 and 23-24 disclose a many-to-one relationship between data line length and cache line length, these claims are not anticipated by Sachs. For at least the same reasons, claims 18-21, which depend from claim 1, and also claims 27-29, which depend from claims 17, 23 and 24, are not anticipated by Sachs. The Examiner's rejection of these claims should be reversed.

Scales Fails to Teach Every Element of Claims 17-21, 23 and 27-28.

The Examiner rejected claims 17-21, 23 and 27-28 under 35 U.S.C. § 102(b) as anticipated by Scales. Because Scales does not disclose each and every element of these pending claims, the Examiner's rejection must be reversed.

Scales describes a bus master that is able to transfer additional operands from main memory in order to fill a single cache *entry* (not a cache *line*) when a requested operand is either improperly aligned or when the requested operand is smaller than the size of a cache entry. This condition may happen when, for example, a bus master is asked by a processor to transfer an 8-bit operand from main memory, but cache entries are sized to handle 32-bit values. In cases like this, where a requested operand is insufficient to fill a single cache entry, Scales describes how a bus master would coordinate the successive transfer of adjacent operands from main memory to fill the 32-bit cache entry. Scales, Abstract; col. 1:36-57.

Because Scales is concerned with cache *entries* and not cache *lines*, anticipate the claimed invention.

Claims 17-21, 23 and 27-28 are Allowable Over Scales.

Claims 17-21, 23 and 27-28 of the present invention are all directed to posting a sequence of external transactions to *fill an entire cache* ***line***, wherein each cache line is sized to store multiple data line lengths of data. According to claim 17, for example, whenever a data request misses the cache, a sequence of external transactions are posted to *fill a cache* ***line*** with data associated with the data request. Scales fails to teach or suggest a method of posting a series of external transactions to fill an entire cache ***line*** with data. On the contrary, Scales merely describes a technique for transferring additional operands to fill a single cache ***entry*** (not an entire cache *line*) whenever an initially-requested operand is too small or is improperly aligned relative to the corresponding cache entry. Scales, col. 1:36-38 ("it is an object of the present invention to provide a bus master which *selectively* attempts to fill entire *entries* in a cache *line*"); col. 1:45-47 (emphasis added); col. 2:63.

The Examiner, presumably not appreciating the difference between a cache *entry* and a cache *line*, rejected claims 17-21, 23 and 27-28 as being anticipated by Scales. Final Office Action (paper no. 18), p. 6 (citing to Scales' discussion of background art, which does not describe filling an entire cache *line*, but instead discusses the general problem of storing partial operands in a single cache *entry*); p. 7-8 (rejecting dependent claims on other grounds). Appellants respectfully maintain that Scales fails to teach or suggest filling a cache ***line***, as described by claims 17-21, 23 and 27-28.

Scales is primarily concerned with supplementing a prior memory request that is either insufficient in size or alignment to fill an entire cache entry. When a memory request fails to fill a cache entry, Scales issues additional memory requests, which are designed to fill the cache entry

with the data that was "missing" from the initial memory request. To accomplish this goal, Scales gives several detailed examples. See Scales, col. 4 – col. 7. Notably, not one of Scales' examples describes a situation in which an entire cache *line* is filled with data associated with an initial data request. Indeed, the Examiner has provided no such reference. Scales is concerned with filling cache entries and not cache lines. Accordingly, Scales has no need to discuss either data lines (the maximum amount that a data bus may transfer) or to discuss filling entire cache lines associated with a memory request. For at least these reasons, claims 17-21, 23 and 27-28 are not anticipated by Scales. The Examiner's rejection of these claims should be reversed.

Sachs and Scales Together Do Not Teach or Suggest Every Element of the Claims

The Examiner rejected claims 1-7, 11-16 and 25-26 under 35 U.S.C. § 103(a) as being obvious over Sachs in view of Scales. Because these references, even if combined, fail to teach or suggest all of the elements of the pending claims, the Examiner's rejection must be reversed.

Each of claims 1-7, 11-16 and 25-26 describe an internal cache, in which each cache line is sized to store multiple data line lengths of data. As the Examiner admits, Sachs fails to teach this feature of the claims. Final Office Action (paper no. 18), p. 9. However, the Examiner suggests that it would have been obvious to one skilled in the art to combine Sachs with Scales, and that the resulting combination would produce the claimed invention. Appellants respectfully maintain that even if Sachs were combined with Scales in the manner suggested by the Examiner, the combination would still fail to describe a cache line that is sized to store multiple data line lengths of data. Additionally, because there is no motivation to combine the two references, their combination is not obvious and the Examiner's rejection must be reversed.

The Combination Fails to Describe the Claimed Invention.

First assume, for the sake of argument, that one of ordinary skill in the art could find some motivation to combine Scales with Sachs. The resulting combination would still fail to describe a cache line that is sized to store multiple data line lengths of data.

Recall that Sachs describes a cache memory structure that stores multiple data words in the same cache line, but which exhibits *less than* one-to-one ratio of data line length to cache line length. Because Sachs fails to describe the many-to-one ratio as claimed by the present invention, Sachs alone does not describe a cache line that stores multiple data line lengths of data.

Scales does not provide what Sachs needs to achieve the present invention. Rather, Scales merely explains how a bus master may be designed to issue additional memory requests to fill a single cache entry when the size or alignment of a memory request does not match the size or

alignment of the target cache entry. While Scales teaches a cache line having multiple cache entries, Scales is only concerned with data transfers that are smaller than the width of the data bus. Thus, instead of teaching a specific ratio of data line length to cache line length, Scales focuses on a limited use of a data bus to transfer only portions of a data item to a single cache entry. Scales, col. 2:28-32. The logical result of the combination of Sachs and Scales, therefore, fails to teach or suggest a cache line that is sized to store multiple data line lengths of data. Accordingly, the Appellants respectfully submit that the obvious rejection to claims 1-7, 11-16 and 25-26 should be reversed.

No Motivation to Combine.

Scales does not provide anything that Sachs needs. For at least this reason, there is no motivation to combine the references.

Recall again that Sachs describes a cache memory structure that stores multiple data words in the same cache line, but which exhibits a *less than* one-to-one ratio of data line length to cache line length, rather than the many-to-one ratio identified in claim 1 of the present invention. Recall also that Scales explains how a bus master may be designed to issue additional memory requests to fill a single cache *entry* when the size or alignment of a prior memory request prevented the returned data from filling the entire cache entry. Thus, Scales is useful in systems that issue memory requests which return data insufficient to fill a single cache entry. As Scales explains, this condition may occur either when the returned data is smaller in size than a cache entry, or when the returned data partially spans two adjoining cache entries. Scales, col. 1:40-47.

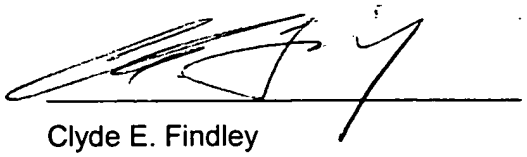
Neither of these conditions occurs in Sachs. Rather, when Sachs issues a memory request for a byte or half-word (both of which are smaller than a single cache entry), it automatically transforms that request into a request for a quadword, which fills the entire cache line. See, e.g., Sachs, col. 26:42; col. 26:61; col. 27:30-31. Sachs structures memory requests in this fashion in order to achieve faster overall performance of cache memory. See, e.g., Sachs, col. 23:12:17 (explaining that quadword output from cache is intended to accelerate cache access time). Because Sachs never issues actual memory requests for data items that are smaller than a cache line, it has no need for the cache entry-filling capabilities which Scales affords. In short, the combination produces no benefit, for it adds nothing that Sachs does not supply alone. Therefore, one of ordinary skill in the art of computer memory system design would have no motivation to combine Scales with Sachs.

SUMMARY

In view of the above, the Appellants respectfully submit that all claims on appeal distinguish over the cited references. The Examiner's rejections of these claims should be reversed. The Appellants respectfully move this Board to reverse the Examiner's decision rejecting claims 1-7, 11-21, and 23-29.

Respectfully submitted,

Date: Nov. 26, 2002



Clyde E. Findley
Reg. No. 50,724

KENYON & KENYON
1500 K Street, NW
Washington, DC 20005
Phone: (202) 220-4200
Facsimile: (202) 220-4201

Appendix: Claims on Appeal

(Brief of Appellants Prudvi, et al., U.S. Patent Application Serial No. 09/212,291)

1. A processing agent to transfer data of a predetermined data line length in an external transaction, the agent comprising an internal cache having a plurality of cache entries, each entry sized to store multiple data line lengths of data.

2. The processing agent of claim 1, wherein the cache entries include a tag portion adapted to store address information.

3. The processing agent of claim 2, wherein the internal cache further comprises
match detection logic for the tag portions, and
control logic provided in communication with the match detection logic.

4. The processing agent of claim 1, wherein the cache entries include a cache coherency state field in association with each data line length of data.

5. The agent of claim 1, further comprising a transaction queue having a plurality of queue entries, the queue entries including a primary entry adapted to store address information and status information of a first external transaction and a secondary entry adapted to store status information of a second external transaction.

6. The agent of claim 5, wherein the status information of the first external transaction includes a field representing whether the first external transaction is part of a multiple transaction sequence.

7. The agent of claim 5, wherein the total number of primary and secondary entries equals the multiple number of data line lengths provided in the cache entries.

11. A processing agent, comprising:
an internal cache having cache entries each sized to store multiple data lines, and
a transaction queue system to post external transactions, each external transaction related to a single data line,
wherein the internal cache and the transaction queue system each receive data requests on a common input.

12. The processing agent of claim 11, wherein the internal cache and the transaction queue system communicate by signal lines.

13. The processing agent of claim 12, wherein the signals lines include a cache hit signal line and a tag hit signal line.

14. The processing agent of claim 11, wherein the transaction queue system comprises a plurality of queue entries, each queue entry comprising:

a primary entry including an address portion and status portion, the status portion provided for a first external transaction of the agent, and

a secondary entry including a status portion provided for a second external transaction.

15. The transaction queue of claim 14, wherein the status portion of the primary entry includes a field representing whether the first transaction is part of a multiple transaction sequence.

16. The transaction queue of claim 14, further comprising control logic adapted to cycle through the queue entries and post transactions therefrom.

17. A method of processing a data request within a processing agent comprising:

posting the data request internally within the agent,

determining whether the request hit the cache,

when the request misses the cache, posting a sequence of external transactions to fill a cache line with data associated with the data request;

wherein each cache line is sized to store multiple data line lengths of data.

18. The method of claim 17, wherein the determining step includes:

comparing address information of the data request with tags stored in the internal cache, and

identifying a cache miss when the address information does not match any stored tag.

19. The method of claim 18, wherein the determining step further includes:

when address information matches a stored tag, reading cache coherency state information associated with the requested data, and

identifying a cache miss when the cache coherency state information is invalid for a request type of the data request.

20. The method of claim 17, further comprising, when the request hits the cache:

determining whether the request hits a tag stored in the cache, and

if so, generating a single external transaction to read the requested data into the agent.

21. The method of claim 20, wherein the second determining step includes:

comparing address information of the data request with tags stored in the internal cache, and

identifying a tag hit when the address information matches a stored tag.

23. A method of processing a data request within a processing agent comprising:

posting the data request internally within the agent,

determining whether the request hit the cache,

when the request misses the cache, posting a series of external transactions from within the agent to fill a cache line with data associated with the data request, the external transactions directed to a data-line-sized data item identified by an address of the data request and to at least one other data-line-sized data item adjacent to the first data item;

wherein each cache line is sized to store multiple data line lengths of data.

24. A processing agent, comprising:

an internal cache having a plurality of cache lines, each cache line including:

a tag portion storing address information, and

a plurality of cache entries, each cache entry sized to store one data line length of data;

wherein the processing agent, in response to a multiple transaction signal, posts a series of external transactions related to the address information, each of said external transactions filling one of said cache entries in the cache line.

25. The processing agent of claim 1, wherein "data line" refers to the maximum amount of data that can be transferred in a single bus transaction.

26. The processing agent of claim 11, wherein "data line" corresponds to the maximum amount of data that can be transferred in a single bus transaction.

27. The method of claim 17, wherein "data line" corresponds to the maximum amount of data that can be transferred in a single bus transaction.

28. The method of claim 23, wherein "data line" corresponds to the maximum amount of data that can be transferred in a single bus transaction.

29. The processing agent of claim 24, wherein "data line" corresponds to the maximum amount of data that can be transferred in a single bus transaction.